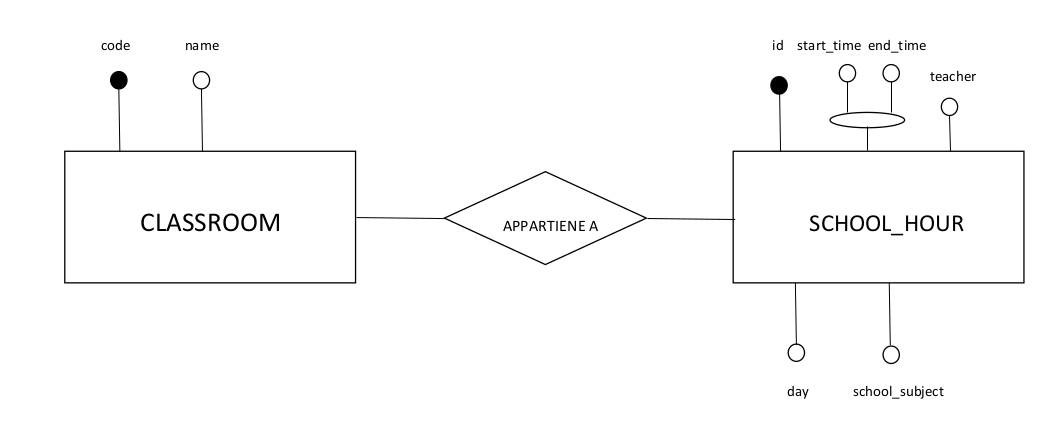
**Diario di lavoro**

|  |  |
| --- | --- |
| Luogo | Scuola d'Arti e Mestieri di Trevano, Canobbio |
| Data | 14 Febbraio 2020 |

Lavori svolti

Lorenzo: ho inserito il sito web sviluppato da Julian nel web server; riorganizzato un po' il codice; aggiunto lo scanner AR e un server di reverse proxy HTTPS con Nginx (vedere *problemi* per il motivo). Ho preso a configurazione del server proxy da questa pagina: <https://linuxtechlab.com/simple-guide-to-configure-nginx-reverse-proxy-with-ssl/>.

Julian: oggi mi sono focalizzato sulla creazione del database. Sono partito con la progettazione dello schema Entità-Relazione per capire bene la logica e il funzionamento del sistema.

Una volta capito bene la struttura ho iniziato a scrivere il codice SQL per la creazione effettiva.

Pierpaolo e Claudio: abbiamo continuato ad estrapolare i dati dalla tabella elenco dell'orario scolastico e abbiamo create un'applicazione con Flask. Ogni volta che testiamo, lo scrapper apre il browser Google Chrome andando sul sito dell'orario della scuola ed inseguito esegue i vari test. Questo possiamo solo lasciarlo in fase di sviluppo, ma certamente per il sistema finale la "GUI" dobbiamo nasconderla.

Per farlo dobbiamo importare *Options*, che si trova direttamente sotto *webdriver* installato in precedenza:

from selenium.webdriver.chrome.options import Options

Per farlo, aggiungere queste righe che permettono di nascondere la "GUI" del browser:

opts = Options()

opts.set\_headless()

asser opts.headless

Aggiungere un parametro a *webdriver.Chrome()*:

self.browser = webdriver.Chrome()

diventa

self.browser = webdriver.Chrome(options=opts)

.

Problemi riscontrati e soluzioni adottate

Lorenzo: quando ho messo il prototipo AR sul sito, ricevevo un errore "Access this website over HTTPS to enter VR mode and grant access to the device sensors.". Ho risolto il problema mettendo in piedi un reverse proxy usando Nginx e dei certificati TLS generati da me con il comando "openssl req -new -newkey rsa:4096 -x509 -sha256 -days 365 -nodes -out cas.crt -keyout cas.key" (i certificati sono probabilmente temporanei).

Julian: ho riscontrato dei piccoli problemi di logica per quanto riguarda le relazioni fra le tabelle ma ho risolto velocemente. Alla fine sono arrivato alla seguente soluzione: [ttps://github.com/lorenzoap/classroom\_ar\_scanner/blob/master/Materiale/Classroom\_Scanner\_AR\_2020\_02\_14/SchoolHours.sql](https://github.com/lorenzoap/classroom_ar_scanner/blob/master/Materiale/Classroom_Scanner_AR_2020_02_14/SchoolHours.sql)

Una volta creato il database e testato il suo funzionamento inserendo dei valori di prova ho cominciato a provare la funzionalità del nostro sito. Assieme a Lorenzo abbiamo messo online il sito grazie a Docker e abbiamo testato la compatibilità su vari dispositivi smartphone. Una volta testato questo ho iniziato a pensare allo sviluppo della funzione GPS. Ho notato che i mezzi a disposizione sono imprecisi e la qualità finale del prodotto non sarà ottimale, ma cercherò di rendere il tutto il più preciso possibile.

Pierpaolo e Claudio:

1) Con Beautifulsoup riuscivamo a ricavare i dati dell'orario scolastico, ma il problema era come filtrarli. Per farlo abbiamo creato un dizionario che ad ogni riga (cioè ad ogni *<td>* della lezione) aggiunge gli elementi corrispondenti. Ad esempio se alla prima riga c'era l'orario "13h15 - 15h45", esso viene aggiunto al dizionario ella voce *time*.

Struttura dizionario:

row\_data = {"time": None, "lesson": None, "teacher": None, "Classe": None}

Struttura parse dati:

1. Ciclo righe della tabella

2. Creazione dizionario

3. Ciclo colonne della tabella

4. Diverse if per ogni caso:

\* Colonna 2: ora lezione

\* Colonna 3: nome lezione

\* Colonna 4: docente

\* Colonna 5: classe

2) Abbiamo avuto problemi quando abbiamo deciso di non usare più la GUI del browser. Siccome i test venivano eseguiti non più con la GUI, abbiamo avuto problemi con i key press simulati. All'inizio i comandi erano semplicemente dei comandi dalla tastiera inviati direttamente al sistema operativo. Prima con GUI: sleep 1s => press DOWN =>sleep 1s => release DOWN => press ENTER => release ENTER. Abbiamo cercato e abbiamo visto che Selenium aveva degli eventi della tastiera ma anche questi non funzionavano. Alla fine abbiamo analizzato il codice sorgente dell'orario e ci siamo resi conto che le sezioni del menu a tendina dal quale si può scegliere il tipo di visualizzazione avevano ognuno un ID. L'ID che corrisponde alla visualizzazione "in elenco" è il seguente: "GInterface.Instances[0].Instances[2]\_1". Quindi grazie ad un semplice test di Selenium abbiamo potuto cercare quell'elemento e fare un semplice click con il mouse virtuale:

self.browser.find\_element\_by\_id("GInterface.Instances[0].Instances[2]\_1").click()

3) Durante il nostro lavoro Lorenzo ci ha posto un grande problema. In effetti voleva che i dati che recupera lo scrapper fossero della settimana successiva a quella corrente. Prima di iniziare a creare la funzione, Pierpaolo ha analizzato il codice sorgente e si è reso conto che la barra delle settimane in cima al calendario era una tabella che conteneva dei *<td>* che a loro volta contenevano delle *<div>* (elemento che esegue l'evento click del mouse e altri eventi). Ogni *<td>* della tabella ha degli ID strani ma con particolarità: ad esempio il *<td>* della tabella della settimana 33 (seconda settimana di agosto) ha l'ID "GInterface.Instances[1].Instances[4]\_j\_1" e continua così incrementando fino alla prima settimana di agosto (settimana 32) con l'ID "GInterface.Instances[1].Instances[4]\_j\_52".

Questo ci ha causato un grande problema, ma Pierpaolo ha avuto la seguente idea:

1. Ricavare il numero della settimana corrente

2. Ciclare ogni elemento *<div>* della tabella della settimana

3. Fare un *if* se il testo che contiene la *<div>* corrisponde al numero della settimana successiva => esegue un click su quell'elemento.

Certamente dobbialo lavorare con dei testi Selenium per cercare elementi.

Particolare funzione che abbiamo trovato per ricavare il numero della settimana:

numero\_settimana = ((datetime.datetime.now() - datetime.datetime(datetime.datetime.now().year, 1, 1)).days // 7) + 1

oppure

numero\_settimana = datetime.datetime.now().strftime("%w")

Dove *strftime* è un metodo per formattare la data (https://programiz.com/python-programming/datetime/strftime)

4) Analizzando il codice, Claudio si è reso conto che all'interno dei loop dei dati avvenivano troppi cicli (a volte più di 157), anche se dovevano risultare di meno visto che non c'erano 157 righe. Oltre ai cicli, venivano aggiunti al dizionario elementi inutili. Per risolverlo abbiamo modificato alcune cose che devono essere inserite prima di scrivere il ciclo delle colonne:

children = row.find\_all("td", recursive=True)

Questo permette di trovare tutti i figli che si trovano nel *<td>* padre (che ha come classe "Grass"). Abbiamo perciò modificato il ciclo delle colonne in:

for j, child in enumerate(children)

5) Alcune classi hanno contemporaneamente la stessa lezione (ad esempio le classi I2AC e I2BC). Quindi il dato che veniva estrapolato era "SAM I2ACSAM i2BC" e ci serviva separare le due. Per farlo abbiamo creato una nuova funzione che utilizza un'espressione regolare e permette di formattare in modo corretto il dato. Abbiamo usato la seguente espressione regolare:

SAM [IED][1-4][ABCD]{2}

Punto della situazione rispetto alla pianificazione

Siamo nei tempi giusti essendo che abbiamo già terminato il prototipo e stiamo implementando già i server.

Programma di massima per la prossima giornata di lavoro

Lorenzo: risolvere un piccolo problema di "zoom" nella pagina /scan, mettere in piedi il database ed eventualmente aggiungere un'interfaccia admin al sito dalla quale controllare il fetch server.

Julian: continuare con l'implementazione del GPS e cercare di risolvere i vari problemi che abbiamo riscontrato durante il nostro percorso.

Pierpaolo e Claudio: dobbiamo estrapolare la data del giorno in modo da poterla utilizzare come ID per stampare tutti i dati utilizzando Flask.